FYS-STK4155 Project 2: Classification and Regression

Erik Johannes Husom

20/10/2019

Abstract

Test of abstract.

Contents

1 Introduction

The aim of this project

2 Theory

2.1 Logistic regression

2.1.1 Gradient descent

2.2 Artificial neural networks

Artificial neural networks (ANN) is a machine learning technique inspired by the networks of neurons that make up the biological brain. There are several different types of neural networks, and in this project we use a multi-layer perceptron (MLP), where there is one or more *hidden layers* between the input and output layers, and each hidden layer consists of several *neurons*. The signal will only go in one direction, from the input layer to the output layer, which makes this a *feedforward neural network* (FNN). Additionally, all neurons of a layer will be connected to every neuron on the previous layer, which makes it a *fully connected layer*. In this section we will review the essential parts of how a neural network functions.

2.2.1 Information flow and forward feeding

The basic idea of this kind of network is that the input information (i. e. the features/predictors of our data set) y, are passed through all of the neurons in the hidden layers, until it ends up in the output layer. Figure 1 shows an example with three inputs, two hidden layers with four neurons each, and one output. This process is called *forward feeding* of the network. Each input is multiplied by a weight w_i when passed into a neuron, and the result is called z_i. The signal is then evaluated by an activation function $a(z_i) = a_i$, which then becomes the output from each neuron. Since we are dealing with several observations, we assemble the weights w_i into a matrix W. Usually we add a bias b to each of the neurons in a hidden layer, to prevent outputs of only zeros. The output a_l of a layer l then becomes (where a_{l-1} is the output from the previous layer):

$$a_l = a(W_a l - 1 + b_l)$$

In the case of the first hidden layer, the input matrix X will take the place of a_{l-1}. The weights are usually initialized with random values, for example using a normal or uniform distribution, because if all the weights were the same, all neurons would give the same output. In this project we use a standard normal distribution when doing classification, but in the case for regression we use the so-called *Xavier initialization* (Glorot and Bengio, n.d.)

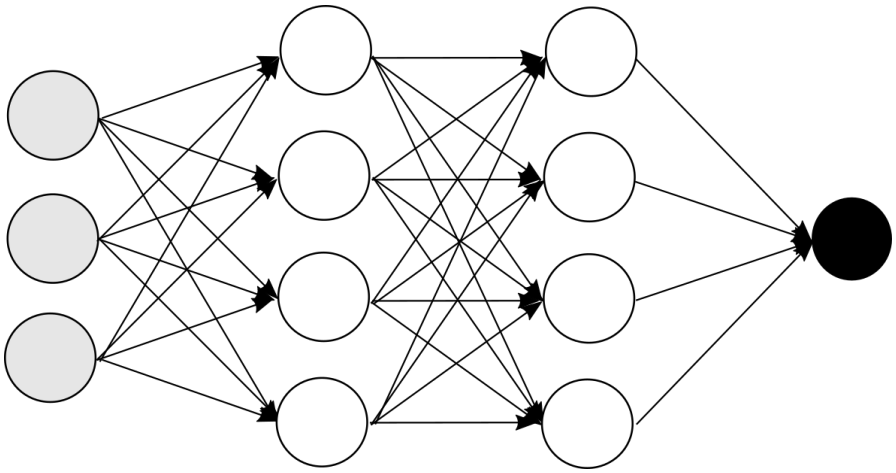

Figure 1: Schematic neural network. The circles represent neurons, and the colors indicate what layer they are a part of: Grey means input layer, white means hidden layer and black means output layer. The arrows show that all neurons of one layer is connected to each of the neurons in the next layer; i. e. we have only fully connected layers.

2.2.2 Activation functions

The choice of activation function may have a huge effect on the model, and there are several options that may work depending on what data set we want to process, how it is scaled and if it is a regression or classification case. In a feedforward neural network, we must have activation functions that are non-constant, bounded, monotonically-increasing and continous for the network to function correctly on complex data sets. It is possible to choose different activation functions for each hidden layer, but in this project we have the same activation function throughout the networks we use.

Common choices for activation functions are the logistic/sigmoid function, the Rectified Linear Unit function (ReLU) and the tanh-function. For classification problems, the sigmoid function, presented in equation (**???**), is often preferred. This function gives only output that is between 0 and 1, which is good when we want to predict the probability as output of the network. In multiclass classification problems the so-called softmax function is a better choice, because it forces the sum of probabilites for the possible classes to be 1, but for binary classification problems, like we have in this project, the sigmoid function is sufficient.

For regression it is more common to use

2.2.3 Output layer and back propagation

After the feedforward process is done, the output layer will contain the predictions of the neural network, which we will compare with the true targets of our training data set. Based in this comparison, we will adjust the weights and biases of the network in order to improve the performance. Since the weights and biases usually are initialized randomly, the first feedforward iteration will most likely give very wrong results. One feedforward pass is usually called an *epoch*, and we choose the number of epochs based on how much "training" the network needs in order to give satisfactory results. The process of adjusting the weights and biases is called *back propagation*.

The comparison between the output from the network $\hat{y}$ and the true targets of our training data set y is done with a predefined cost function $\mathcal{C}$. In our classification problem, we use the

For regression we use a slightly modified version of the mean squared error:

$$\mathcal{C} = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

The factor $\frac{1}{2}$ is used to simplify the derivative of the cost function, which we will need for the back propagation:

$$\text{ReLU} = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

2.2.4 Minibatches

3 Methods

3.1 Minibatching

Results

Glorot, Xavier, and Yoshua Bengio. n.d. "Understanding the Difficulty of Training Deep Feedforward Neural Networks," 8.